ИНФОРМАТИКА, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И УПРАВЛЕНИЕ
INFORMATION TECHNOLOGY, COMPUTER SCIENCE, AND MANAGEMENT



УДК 519.683.4
https://doi.org/10.23947/1992-5980-2019-19-1-86-92

Оптимизация арифметического кодера для сжатия изображений, полученных при дистанционном зондировании водных объектов[*]

Р. В. Арзуманян[1**]
[1] Институт компьютерных технологий и информационной безопасности Южного федерального университета, г. Таганрог, Российская Федерация

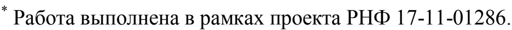
Arithmetic coder optimization for compressing images obtained through remote probing of water bodies [***]

R. V. Arzumanyan[1**]
[1] Institute of Computer Technology and Information Security, Southern Federal University, Taganrog, Russian Federation

Введение. Предложенный в статье быстрый программный алгоритм арифметического кодирования предназначен для сжатия цифровых изображений. Показано, каким образом сложность алгоритма арифметического кодера зависит от критериев сложности (при этом размер входа не учитывается). В процессе работы определены наиболее вычислительно сложные части алгоритма арифметического кодера. Выполнена оптимизация производительности их программной реализации.
Кодеки с новым алгоритмом сжимают без учета межкадровой разницы фото- и видеоматериалы, полученные при дистанционном зондировании водных объектов.
Материалы и методы. В представленной научной работе использована подборка спутниковых снимков акватории Азовского моря. При этом оптимизирован программный алгоритм арифметического кодера, проведено теоретическое исследование, выполнен вычислительный эксперимент.
Результаты исследования. Увеличена производительность программной реализации арифметического кодера на примере видеокодека VP9. Для измерения времени выполнения произведены многочисленные запуски эталонного и модифицированного кодеков. Сравнение среднего времени их исполнения показало, что производительность модифицированного кодека на 5,21 % выше. Прирост общей производительности для арифметического декодирования составил 7,33 %.
Обсуждение и заключения. Увеличение скорости работы новейших алгоритмов сжатия цифровых фото- и видеоизображений позволяет применять их на мобильных вычислительных платформах, в том числе в составе бортовой электроники беспилотных летательных аппаратов. Теоретические результаты данной работы расширяют методы анализа сложности алгоритма в среднем случае. Они могут использоваться в ситуации, когда количество шагов алгоритма зависит не только от размеров входа, но и от неизмеримых критериев (например, от схемы обращения к общей оперативной памяти со стороны параллельных процессоров).

Introduction. The fast program algorithm of arithmetic coding proposed in the paper is for the compression of digital images. It is shown how the complexity of the arithmetic coder algorithm depends on the complexity measures (the input size is not considered). In the course of work, the most computationally complex parts of the arithmetic coder algorithm are determined. Performance optimization of their software implementation is carried out. Codecs with the new algorithm compress photo and video records obtained through the remote probing of water bodies without frame-to-frame difference.
Materials and Methods. In the presented paper, a selection of satellite images of the Azov Sea area was used. At this, the software algorithm of the arithmetic coder was optimized; a theoretical study was conducted; and a computational experiment was performed.
Research Results. The performance of the software implementation of the arithmetic coder is increased by the example of the VP9 video codec. Numerous launches of reference and modified codecs were made to measure the runtime. Comparison of the average time of their execution showed that the modified codec performance is 5.21% higher. The overall performance improvement for arithmetic decoding was 7.33%.
Discussion and Conclusions. Increase in the speed of the latest digital photo and video image compression algorithms allows them to be used on mobile computing platforms, also as part of the onboard electronics of unmanned aerial vehicles. The theoretical results of this work extend tools of the average-case complexity analysis of the algorithm. They can be used in case where the number of algorithm steps depends not only on the input size, but also on non-measurable criteria (for example, on the common RAM access scheme from parallel processors).





[*] Работа выполнена в рамках проекта РНФ 17-11-01286.
[**] E-mail: roman.arzum@gmail.com
[***] The research is done within the frame of RSF project no. 17-11-01286.

Ключевые слова: арифметическое кодирование,оптимизация производительности, сжатие изображений, сложность алгоритма в среднем, видеокодек.

Keywords: arithmetical coding, performance optimiza tion, image compression, average-case algorithm complexity, video codec.

***Образец для цитирования*:** Арзуманян, Р. В. Оптимизация арифметического кодера для сжатия изображений, полученных при дистанционном зондировании водных объектов / Р. В. Арзуманян // Вестник Донского гос. техн. ун-та. — 2019. — Т. 19, № 1. — С. 86–92. https://doi.org/10.23947/1992-5980-2019-19-1-86-92

For citation: R.V. Arzumanyan. Arithmetic coder optimization for compressing images obtained through remote probing of water bodies. Vestnik of DSTU, 2019, vol. 19, no. 1, pp. 86–92. https://doi.org/10.23947/1992-5980-2019-19-1-86-92

Введение. Мониторинг состояния акватории зачастую проводят с помощью беспилотных летательных аппаратов (БПЛА), ведущих аэрофотосъемку в видимом и инфракрасном диапазонах. Технические возможности мобильной камеры, которой оснащен БПЛА, накладывают ряд ограничений на оборудование, обрабатывающее и хранящее снятый материал до возвращения БПЛА. В частности, необходимо учитывать перечисленные ниже факторы.

1. Энергоэффективность оборудования, которое кодирует отснятый материал, поскольку от этого напрямую зависит продолжительность автономной работы БПЛА.
2. Эффективность сжатия фото- и видеоданных во время полета. Изображения в высоком разрешении занимают значительный объем памяти программного обеспечения (ПО), и это ограничивает количество информации, которую может накопить БПЛА.

Практически вся аппаратура для фото- и видеосъемки аппаратно поддерживает наиболее распространенный кодек для сжатия изображений JPEG. Однако он уступает наиболее современным кодекам HEVC и VP9, которые не только поддерживают видеопоследовательности, но и позволяют лучше сжимать отдельные изображения. Так, GoogleVP9 демонстрирует аналогичное JPEG визуальное качество по метрике SSIM (структурная схожесть) и при этом сжимает изображения на 25–34 % сильнее [1]. В сравнении с тем же JPEG кодек HEVC позволяет улучшить степень сжатия на 10–44 % по метрике PSNR (пиковое соотношение сигнала к шуму) [2]. Однако следствием высокой степени сжатия при меньшем размере битового потока является большая вычислительная сложность кодеков HEVC и VP9 [3, 4]. Архитектура кодека JPEG в общем виде представлена на рис 1.

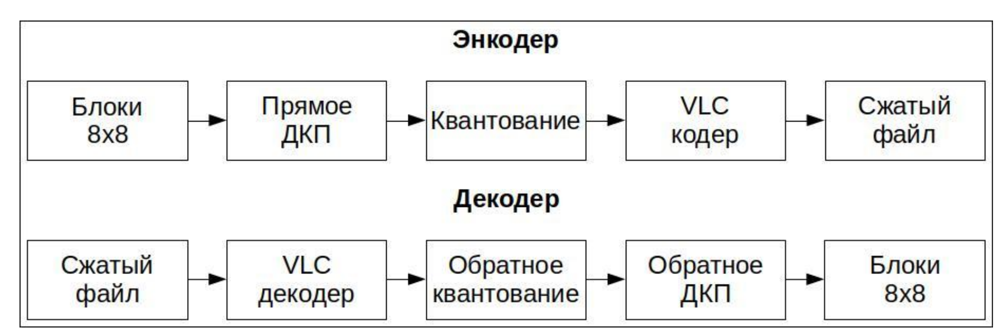


Рис. 1. Блок-схема работы кодека JPEG

Здесь под VLC понимается сжатие кодами переменной длины (variable length coding). Входной кадр разбивается на блоки фиксированного размера 8×8. Каждый из них подвергается прямому дискретному косинусному преобразованию (ДКП), квантизации коэффициентов преобразования и последующему энтропийному сжатию при помощи алгоритма Хаффмана [4]. Дискретное косинусное преобразование выполняется в целочисленном виде [5]. С момента принятия стандарта JPEG в 1992 году разработано множество быстрых алгоритмов, которые позволяют провести преобразование целиком в регистрах центрального процессора. Энтропийное сжатие Хаффмана также не является вычислительно сложной задачей, поэтому даже мобильные процессоры в программном режиме выполняют сжатие и декодирование изображений JPEG [6].

HEVC [6] и VP9 принципиально схожи и представляют собой гибридные блочные кодеки с разбиением кадра на блоки переменного размера, внутрикадровым предсказанием, дискретным преобразованием и последующей фильтрацией для устранения артефактов блочности. Блок-схема работы кодека HEVC представлена на рис. 2.

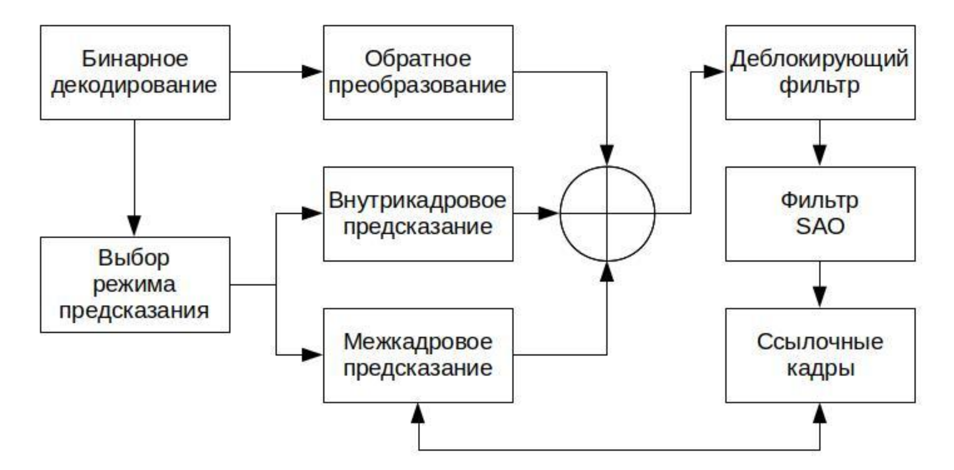


Рис. 2. Блок-схема кодека HEVC

Здесь аббревиатурой SAO обозначен фильтр с адаптивным смещением (sample adaptive offset). Помимо перечисленных алгоритмов реконструкции изображения в обоих кодеках применяется контекстно-адаптивное двоичное арифметическое энтропийное кодирование, значительно более сложное, чем сжатие Хаффмана. При высоком уровне визуального качества именно арифметическое кодирование занимает значительную часть общего времени работы декодера. В общем виде схема работы арифметического кодера представлена на рис. 3.

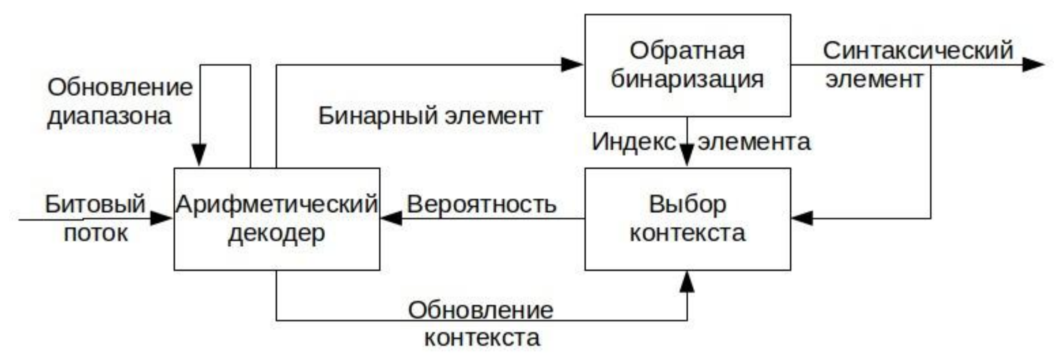


Рис. 3. Блок-схема арифметического кодера

Кодеки нового поколения предполагается использовать в том числе для сжатия изображений, полученных при дистанционном зондировании природных объектов. В этом случае необходимо оптимизировать работу гибридных блочных кодеков для обеспечения обработки фото- и видеоданных БПЛА в режиме реального времени. Необходимо уделить особое внимание оптимизации двоичного контекстно-адаптивного арифметического кодирования, ввиду его существенной вычислительной сложности.

Цель данного исследования — ускорить работу арифметического декодера на мобильных процессорах архитектуры ARM. Это позволит повысить быстродействие видеокодеков Google VP9 и применять их в бортовой электронике БПЛА для дистанционного зондирования водных объектов. Применение улучшенных инструментов компрессии фото и видео, полученных в результате аэрофотосъемки, позволяет увеличить объем сохраняемых данных, повысить их визуальное качество и разрешение.

Основная часть. К основным составляющим современного видеокодека относятся бинаризация синтаксических элементов битового потока и адаптивное бинарное кодирование этих элементов в битовый поток. Данная стадия не подлежит векторизации и распараллеливанию, но может быть оптимизирована путем статистического анализа входных данных. Основной способ прогнозирования времени работы программы — анализ сложности соответствующего алгоритма. Выделяют сложность в лучшем, худшем случае и сложность в среднем. $\sqsupset x$ — входные данные алгоритма A, по которым вычисляется выход алгоритма y. Функцию затрат алгоритма по времени обозначим $C_A^T(x)$, а функцию затрат по памяти — $C_A^S(x)$. Временной и пространственной сложностью A в худшем случае будем называть функции числового аргумента

$$T_A(n) = \max_{\|x\|=n} C_A^T(x),$$
$$S_A(n) = \max_{\|x\|=n} C_A^S(x).$$

Рассмотрим конечное множество входов размера n:

$$X_n = \{x: \|x\| = n\}.$$

$\forall x \in X_n$ соответствует вероятность:

$$P_n(x) \in [0,1]: \sum_{x \in X_n} P_n(x) = 1.$$

Сложностью в среднем называют математическое ожидание:

$$T_A = \sum_{x \in X_n} P_n(x) C_A^T(x),$$
$$S_A = \sum_{x \in X_n} P_n(x) C_A^S(x).$$

Описанный подход является классическим для анализа сложности алгоритма в среднем и подробно описан в [7, 8]. Отметим причины, по которым применение данного метода на практике может быть затруднено или нецелесообразно.

1. Разница между числом шагов алгоритма в теории и количеством тактов процессора, необходимым для выполнения шага на практике. Так, большинство современных центральных процессоров производят сложение и умножение за один такт, в то время как остаток от деления вычисляется за десятки тактов.
2. Аппаратные особенности системы памяти. В современных компьютерах применяется многоуровневая иерархия памяти. Ее компоненты работают на разных скоростях. Обращения к памяти занимают значительно больше времени, чем операции в регистрах.
3. Оптимизирующие компиляторы и аппаратные планировщики. При сборке исполняемых файлов оптимизирующие компиляторы значительно преобразовывают код, не меняя автомат состояний программы. Аппаратные планировщики центрального процессора изменяют порядок исполнения инструкций для большей производительности и предсказывают условные переходы, а контроллеры кэша производят чтение из памяти блоками.
4. В случае программной реализации алгоритмов на процессорах общего назначения программные компоненты взаимно влияют друг на друга. Например, планировщик задач разделяет процессорное время, и параллельные процессы, имеющие несколько потоков, могут выполняться на переменном числе ядер процессора.

Предлагаемая модификация метода анализа сложности алгоритма служит теоретическим дополнением для практических инструментов измерения производительности — таких, например, как профилирование и инструментация кода программы. Впервые метод разбиения входов алгоритма на классы сложности был представлен в [9]. Рассмотрим алгоритм A и множество всех возможных входных данных:

$$G: \{g_1, g_2 \dots\},$$

А также все возможные выборки из G, различные по размеру и составу:

$$g_i: \{g_i^1, g_i^2 \dots\}.$$

Множество критериев сложности реализации алгоритма (например, число циклов процессора, время исполнения и т. п.):

$$\propto_i: g_i \to \mathbb{R}.$$

Множество критериев сложности:

$$A: \{\propto_1, \propto_2 \dots\}.$$

Данное множество обладает перечисленными ниже свойствами.

1. $\forall \propto_1, \propto_2 \in A: \propto_1 \neq \propto_2$ — все элементы A различны.
2. $\forall \propto_i \in A$ разбивает G на множество классов эквивалентности по сложности

$$G(\alpha_i) = \{g_i^1 \cap g_i^2 \dots\}.$$

3. Все выборки из $G(\alpha_i)$ имеют одинаковую сложность:

$$\exists\ \alpha_i \in A, g_i^k \in G(\alpha_i), \alpha_i: g_i^k \to r_i, k \in [0, \|G(\alpha_i)\|], r_i \in \mathbb{R}.$$

Итак, все элементы A различны, поэтому их можно переупорядочить так, чтобы функция сложности от критерия была неубывающей на всем множестве критериев. Ожидаемая сложность схожа с оценками сложности алгоритма в среднем для дискретной и непрерывной вероятности сложности. Для дискретного случая:

$$R(A) = \sum_{\alpha_i \in A} r_i p_i,$$

для непрерывного случая:

$$R(A) = \int_A r\, dF(r).$$

Применим рассматриваемый метод для анализа сложности арифметического кодека. Процесс энтропийного сжатия [10, 11] можно разделить на составляющие части.

1. Бинаризация, или преобразование кодируемого символа (синтаксического элемента сжатого битового потока) в строку, состоящую из нулей и единиц (бинарную строку).

2. Моделирование контекста для сжатия синтаксических элементов в обычном режиме. Для синтаксических элементов, чье статистическое распределение близко к нормальному, данный этап не выполняется, и они кодируются в режиме обхода (bypass).

3. Арифметическое кодирование бинарной строки.

Рассмотрим более подробно схему арифметического декодирования кодека Google VP9, а именно ту часть, которая связана с субэкспоненциальным кодированием синтаксических элементов.

Представим алгоритм субэкспоненциального кодирования в общем виде [12]. На первом шаге вычисляются значения переменных:

$$b = \begin{cases} k: n < 2^k \\ \lfloor \log_2 n \rfloor : n \geq 2^k, \end{cases}$$

$$u = \begin{cases} 0: n < 2^k \\ b - k + 1: n \geq 2^k, \end{cases}$$

где k — параметрическое значение, для кодека Google VP9 оно равно 4.

На втором шаге унарный код $u(u+1)$ бит дополняется младшими битами n. Длина кода равна:

$$u + 1 + n = \begin{cases} k + 1: n < 2^k \\ 2\lfloor \log_2 n \rfloor - k + 2: n \geq 2^k. \end{cases}$$

Таким образом, декодирование литерала сводится к декодированию составляющих его бит в цикле. Для оптимизации производительности данного алгоритма важно знать распределение вероятности длин литералов. Литералы, занимающие в сжатом битовом потоке наибольшее количество бит (такие, как коэффициенты обратного преобразования и вектора движения), кодируются сериями, поэтому высока вероятность, что в сжатом битовом потоке распределение длин литералов будет вырожденным с многочисленными повторами элементов с одинаковым значением. Для проверки этой гипотезы собраны экспериментальные данные о распределении длин литералов на наборе спутниковых снимков акватории Азовского моря (табл. 1).

Таблица 1

Длина литерала, бит	1	2	3	4	5	6
Вероятность, %	0,94	0	67,35	18,25	0	13,46

Наиболее вероятными являются литералы длиной 3, 4 и 6 бит. Максимальная возможная длина литерала для данной последовательности составляет всего 6 бит. Этот факт важен для программной оптимизации функции субэкспоненциального декодирования литерала. В рамках оптимизации реального кодека нас в первую очередь интересует критерий времени исполнения. Для получения множества сложностей $R : \{r_0, \dots, r_4\}$ будем профилировать производительность программы. Множество уникальных элементов R составит множество критериев сложности времени исполнения A. На основе полученных данных для оптимизации применены перечисленные ниже подходы.

1. Сохранение результатов вычисления длины литерала для декодирования серий литералов одинаковой длины.
2. Размотка цикла субэкспоненциального декодирования литерала.
3. Более эффективный алгоритм расчета числа бит в литерале.
4. Более эффективное использование регистров процессора непосредственно внутри функции арифметического декодирования.

Реализация пунктов 1 и 4 достаточно очевидна, поэтому рассмотрим более подробно пункты 2 и 3. Внутри функции субэкспоненциального декодирования происходит составление декодированного литерала по битам, декодированным из сжатого битового потока. В данном случае узким местом является цикл с переменным количеством итераций. Его можно заменить набором switch-case без конструкции break в конце. Данная техника известна как метод Даффа (Duff's device). Она позволяет заменить несколько итераций цикла последовательным выполнением инструкций без необходимости условных переходов. Величина битового сдвига — константа, которую не нужно считывать из регистра — счетчика цикла.

Листинг 1: Оригинальная функция декодирования литерала

```
static int vp9_read_literal(vp9_reader *br, int bits)
{
int z = 0, bit;
 for (bit = bits – 1; bit >= 0; bit –)
   z |= vp9_read_bit(br) << bit;
return z;
}
```

Листинг 2: Модифицированная функция декодирования литерала

```
static int vp9_read_literal(vp9_reader *br, int bits) {
 register int z = 0;
 switch(bits -1){
   case 6: z |= vp9_read(br, 128) << 6;
   case 5: z |= vp9_read(br, 128) << 5;
   case 4: z |= vp9_read(br, 128) << 4;
   case 3: z |= vp9_read(br, 128) << 3;
   case 2: z |= vp9_read(br, 128) << 2;
   case 1: z |= vp9_read(br, 128) << 1;
   case 0: z |= vp9_read(br, 128);
break;
}
return z;
}
```

Еще одно узкое место — расчет числа бит литерала в цикле while [13]. Данное решение плохо тем, что число итераций цикла непредсказуемо. Вместо него был использован быстрый алгоритм подсчета бит [14, 15], который выполняет расчет за фиксированное число шагов без использования условных переходов.

Листинг 3: Быстрый подсчет числа бит в литерале

```
Unsig ned intv; // 32-битный аргумент
Register unsig ned intr; // переменная для числа бит
register unsigned int shift;
r = (v > 0xFFFF) << 4;
v >>= r;
shift = (v > 0xFF) << 3;
v >>= shift;
r |= shift;
shift = (v > 0xF) << 2;
v >>= shift;
r |= shift;
shift = (v > 0x3) << 1;
v >>= shift;
r |= shift;
r |= (v >> 1);
```

Для измерения времени выполнения произведены многочисленные запуски эталонного и модифицированного кодеков. При этом сравнивалось среднее время их исполнения. Выяснилось, что производительность модифицированного кодека на 5,21 % выше. Прирост общей производительности для арифметического декодирования составил 7,33 %.

Выводы. Проведена оптимизация работы арифметического кодера в составе видеокодека на примере стандарта Google VP9. Для решения поставленной задачи предложена модификация метода анализа сложности алгоритма в среднем. В основе данного подхода — разбиение множества входов на классы эквивалентности по сложности. Рассмотренный метод позволяет прогнозировать сложность алгоритма в среднем для тех случаев, когда количество шагов алгоритма и время его исполнения зависят от сложно измеримых параметров, что характерно для контекстно-адаптивного арифметического кодирования. Предложенный метод применен для оптимизации скорости работы арифметического двоичного кодера (на примере кодека Google VP9) применительно к задачам сжатия изображений, полученных при дистанционном зондировании водных объектов. Результаты работы позволяют применить передовые методы сжатия фото- и видеоданных, полученных при аэрофотосъемке водных объектов. Таким образом можно увеличить объем накапливаемых данных, повысить визуальное качество и разрешение отснятого материала на 25–34 % (по метрике визуального качества SSIM) и увеличить скорость работы арифметического кодера на 7 %.

Библиографический список

1. WebP Compression Study [Электронный ресурс] / Google Developers. — Режим доступа: https://developers. google.com/ speed/webp/docs/webp_study (дата обращения 01.02.19).

2. Nguyenand, T. Objective Performance Evaluation of the HEVC Main Still Picture Profile / T. Nguyenand, D. Marpe // IEEE Transactions on Circuits and Systems for Video Technology. — 2015. — Vol. 25, № 5. — P. 790–797.

3. Блейхут, Р. Быстрые алгоритмы цифровой обработки сигналов / Р. Блейхут. — Москва : Мир, 1989. — 448 с.

4. Wallace, G. K. The JPEG still picture compression standard / G. K. Wallace // IEEE Transactions on Consumer Electronics. — 1992. — Vol. 38, № 1. — P. XVIII–XXXIV.

5. Дворкович, А. В. Цифровые видеоинформационные системы (теория и практика) / А. В. Дворкович, В. П. Дворкович. — Москва : Техносфера, 2012. — 1009 с.

6. Asaduzzaman, A. Performance-power analysis of H.265/HEVC and H.264/AVC running on multicore cache systems [Электронный ресурс] / A. Asaduzzaman, V. R. Suryanarayana, M. Rahman // Intelligent Signal Processing and Communications Systems. — Режим доступа: https://ieeexplore.ieee.org/document/6704542 (дата обращения 01.02.19).

7. Sedgewick, R. Algorithms. Fourth edition / R. Sedgewick, K. Wayne. — Upper Saddle River : Addison-Wesley, 2016. — 960 p.

8. Introduction to Algorithms / T. H. Cormen [et al.]. — 3rd edition. — Cambridge ; London : The MIT Press, 2009. — 1296 p.

9. Welch, W. J. Algorithmic complexity: three NP — hard problems in computation all statistics / W. J. Welch // Journal of Statistical Computation and Simulation. — 1982. — Vol. 15, № 1. — P. 17–25.

10. High efficiency video coding [Электронный ресурс] / Fraunhofer Heinrich Hertz Institute. — Режим доступа: http://hevc.info/ (дата обращения: 01.02.19).

11. Sze, V. Parallelization of CABAC transform coefficient coding for HEVC [Электронный ресурс] / V. Sze, M. Budagavi // Semantic Scholar / Allen Institute for Artificial Intelligence Logo. — Режим доступа: https://www.semanticscholar.org/paper/Parallelization-of-CABAC-transform-coefficient-for-SzeBudagavi/0653a22ff7b82bdd0130cea8b597a7024ab46882 (дата обращения: 01.02.19).

12. Salomon. D. Handbook of data compression / D. Salomon, G. Motta. — London ; Dordrecht ; Heidelberg ; New York : Springer-Verlag, 2010. — 1360 p.

13. Anderson, S. E. Bit Twiddling Hacks [Электронный ресурс] / S. E. Anderson. — Режим доступа: https://graphics.stanford.edu/~seander/bithacks.html (дата обращения 01.02.19).

14. Гервич, Л. Р. Программирование экзафлопсных систем / Л. Р. Гервич, Б. Я. Штейнберг, М. В. Юрушкин // Открытые системы. СУБД. — 2013. — Т. 8.— С. 26–29.

15. Уоррен-мл., Г. С. Алгоритмические трюки для программистов / Г.-С. Уоррен- мл. — 2-е изд. — Москва : Вильямс, 2013. — 512 с.

Поступила в редакцию 02.11.2018
Сдана в редакцию 03.11.2018
Запланирована в номер 15.01.2019

Received 02.11.2018
Submitted 03.11.2018
Scheduled in the issue 15.01.2019

Об авторе:

Арзуманян Роман Вадимович,
аспирант кафедры «Интеллектуальные многопроцессорные системы» Института компьютерных технологий и информационной безопасности Южного федерального университета (РФ, 347922, Ростовская обл., Таганрог, ул. Чехова, 22),
ORCID: https://orcid.org/0000-0003-3370-5093
roman.arzum@gmail.com

Author:

Arzumanyan, Roman V.,
postgraduate of the Intelligent Multiprocessor Systems Department, Institute of Computer Technology and Information Security, Southern Federal University (22, ul. Chekhova, Taganrog, Rostov Region, 347922, RF),
ORCID: https://orcid.org/0000-0003-3370-5093
roman.arzum@gmail.com